PowerShell v3 命令与虚拟化

PowerShell v3 命令与虚拟化

《PowerShell v3 入门级教程》全面介绍了 PowerShell v3 的新功能、功能应用以及其他管理技术等等。是一本入门级教程，这本《PowerShell v3 命令与虚拟化》技术手册专注 PowerShell v3 在虚拟化工作中的实用功能。之前会介绍基本的 PowerShell 命令、在 Hyper-V 3.0 中的使用等。

PowerShell v3 命令

要想在工作中自如运用 PowerShell v3，首先要掌握基本的 PowerShell 命令。这里详解 PowerShell 命令的基本知识以及常会用到的命令。

- 为什么远程运行 PowerShell 命令容易出错？
- 为什么 PowerShell 变量内容不能进行比较？
- Hyper-V 3.0 的 PowerShell v3：命令行用户的新乐趣
- 练就火眼金睛 了解优良 PowerShell 函数四大特性
- 组策略常用的五大 PowerShell 命令

PowerShell v3 与 Hyper-V 3.0

随着 Windows Server 2012 的面世，PowerShell v3 与 Hyper-V 3.0 大方光彩。看二者如何让你的工作更便捷更出色。

- Hyper-V 3.0：用 PowerShell 输入输出虚拟机
- 用 Hyper-V 3.0 与 PowerShell 管理可扩展交换机

PowerShell v3 虚拟化应用

PowerShell v3 在虚拟化环境中的何实用价值不可小觑。掌握如何使用 PowerShell 管理虚拟桌面、虚拟服务器、热迁移。

- FAQ：使用 Windows PowerShell 命令管理虚拟桌面
- 如何使用 Windows PowerShell cmdlets 进行 XenServer 管理
- 使用 SCVMM PowerShell cmdlets 自定义 Hyper-V 热迁移
- 使用 XenServer PowerShell cmdlets 开发高级脚本

为什么远程运行 PowerShell 命令容易出错

真高兴看到人们使用 PowerShell Remoting 在远程机器上运行命令。这是未来技术的趋势，但是也有点复杂。

只访问本地资源

在开始使用之前，确保命令不访问远程机器上的非本地资源。假如你运行一台 Windows 7 或 Windows 8 电脑，使用 Remoting 将一条命令传输给搭载 Windows Server 2008 或者更新版本的计算机 B，这条命令就能操作计算机 B 上的所有资源。这是不允许的，特别是访问网络。

为什么？这是因为 Remoting 将你的证书委派给计算机 B，但是安全起见，计算机 B 不允许更进一步委派证书。

如果你需要这么做，你可以通过使用 CredSSP 协议实现多级授权。因为有许多安全因素，所以在开始之前，你需要明确。你可以从免费的电子书 PowerShell Remoting 的秘密浏览更多信息，电子书中有一整章节是关于 CredSSP 的。

时间问题

命令的另一个可能性问题在于时间。当你在本地手工运行一条命令时，输入、点击“回车”、阅读结果需要一定的时间。在你开始输入下一条命令时，上一条的命令已经运行完成。

当你发送一批命令到远程机器时，并不一定如此。机器在第一条命令执行完成之间就可以转到下条命令。尤其是在运行启动单独的进程或线程的外部命令时。

为了解决问题，使用 Enter-PSSession 连接到远程机器，并手动运行试图发送的任何命令。如果有用，在发送批量命令时，换成使用 Invoke-Command。否则，你会遇到时间问题。

你有几种解决问题的方法：你可以使用 Start-Sleep 命令暂停 shell；也可以进行必要的检查，如文件是否存在；在执行其他命令之前，将 shell 睡眠并重现检查，然后继续开始。

外部命令问题

还有一种可能性只适用于外部命令，而不是本地 PowerShell cmdlets，那就是远程机器错误解释了你的命令。在 PowerShell v3 中，前面的外部命令使用两个破折号（“--”）告诉 shell 照原来的样子传递给 Cmd.exe，甚至不用解析。这通常用来解决外部命令语法问题。

远程规则

如果你通过远程桌面连接或者从服务器直接登录到远程机器的控制台这种方式来成功运行命令的话，那么这条命令不会通过 PowerShell Remoting 运行，这时你遭遇了“环境问题”。

使用 Invoke-Command、Enter-PSSession 或其他方法远程进入机器以后，不会像登录到控制台或者远程桌面会话那样得到完整的交互式桌面会话。PowerShell 不执行概要文件的脚本，并且没有完整的用户环境。我看到大量的命令都是出于这个原因，但也有可能有一些没有这种问题。

如果你也有这种问题，你可以尝试在远程机器上调度自己的命令。Windows 的任务调度程序拥有一个完整的用户帐户，不能保证能够修复问题，但值得一试。

如果这样也不能解决问题，那就是卡住了。如果你的命令需要一个完整的、交互式桌面环境来运行，那么你就必须按照上述方法运行。

为什么 PowerShell 变量内容不能进行比较？

下面是一个简短的代码片段，演示了最常见的一个 PowerShell 问题：

```
$file = Dir C:\MyDirectory

if ($file.name  -like '*archive*') {

Write-Host "$($file.name) is an archive"

}
```

这段代码的目的是查看大量的文件，并确定文件名是否含有单词“archive”。不幸的是，这个脚本里有一些经典的错误。

首先， PowerShell 中的 Dir 命令有可能返回多个对象以及多种类型的对象。当在文件系统驱动器运行时，返回文件和文件夹。在这种情况下，该脚本将是一个不错的命令，因为两个文件和文件夹有一个 Name 属性（$file.name 引用名称属性），但是这可能并不是你所希望得到的。在 PowerShell v3 中，你可以通过添加 -File 参数限制 Dir 命令只返回文件。阅读帮助命令查看其他你可以指定的限制。

第二， 脚本中的$file 变量通常包含多个对象，因为文件系统中大部分文件夹包含其他文件夹和多个文件。但这并不总是一个问题，因为 PowerShell 能很好地把多个对象放到一个变量中。

下面的比较存在问题：

```
if ($file.name  -like '*archive*') {
```

你打算用哪个文件的 Name 属性同“*archive *”比较？在 PowerShell v3 中，使用 $file.name 是完全合法的。深入了解，PowerShell 将此命令描述成“我想访问$file 变量中所有对象的 Name 属性。”结果仍是多个对象。在这种情况下，结果包含了多个文件和文件夹的名称的字符串对象。

问题在于 PowerShell 的比较运算符，如 - like 运算符，不能处理多个对象。这些运算符设计用于返回 True 与 False 值。如果某些文件命名包含 archive 而其他没有，返回 Maybe 值时，运算符该如何做呢？

解决这个问题的方法是列举文件，因此一次就比较一个。

对于 PowerShell v3，正确方式是：

```
$files = Dir C:\MyDirectory  -File

foreach ($file in $files) {

if ($file.name  -like '*archive*') {

Write-Host "$($file.name) is an archive"

}

}
```

Foreach 架构目的是接收第二文件（$files）中的多个对象，然后每次移动集合中的一个对象。每一次经过，架构从集合中采收一个对象，将其放置在变量的首个专门文件中（例如$file）。以这种方式，你可以每次只运行一个对象。

虽然使用同个单词的单数与复数很常见，比如$file 与$files，但不需要使用 Foreach。这样做可让你的代码更易读，但 PowerShell 不关心这点。

下面代码也有效：

```
$apples = Dir C:\MyDirectory  -File

foreach ($fred in $apples) {

    if ($fred.name  -like '*archive*') {

Write-Host "$($fred.name) is an archive"

}

}
```

在 PowerShell 中，有多种方式做任何事。下面的代码也能实现上面的结果：

```
Dir C:\MyDirectory  -File |
```

```
foreach-object {

if ($_.name  -like '*archive*') {

Write-Host "$($_.name) is an archive"

}

}
```

该种技术使用 ForEach-Object cmdlet 代替 Foreach 脚本架构。这个 cmdlet 使用不同的语法，不用给其具体的变量名称。相反，它给你一个预定的变量（$），每次只包含一个对象。两种技术花费的时间都相同，实现的效果都一样。

你也可以这样做：

```
Dir C:\MyDirectory  -File |

where-object { $_.name  -like '*archive*' } |

foreach-object {

Write-Host "$($_.name) is an archive"

}
```

这使用了命令行技术而不是脚本类型的技术，但也完全有效。

学习 PowerShell 最困难的地方在于任何任务都有太多有效方式可选。如果你要成功地将其他例子改头换面，貌似你需要懂得所有的实现方式。这种多样性也意味着你能采用你熟悉的方式，平衡你给环境带来的现有技巧。

了解我的人可能疑惑为啥我这么唠叨，在本文中碎碎念 Write-Host 的使用案例。这绝对不是从脚本生产输出的最佳方式，当然在以后文章中我会详解的哦。

Hyper-V 3.0 的 PowerShell v3：命令行用户的新乐趣

随着 Hyper-V 3.0 和 Windows Server 8 即将发布，微软进一步明确了 PowerShell v3 将成为其服务器平台底层管理标准。最新的 PowerShell v3 包含了大量全新的 cmdlets，实现服务器和 Windows 8 桌面的管理和自动化，其中也包括有超过 1000 条 cmdlets 的 Hyper-V。

在过去几年中，微软增加了 PowerShell 脚本语言的戏码，通过它搭建很多产品的管理体系，最为著名的就是 Exchange 2010。而即将出现的 PowerShell v3 将作为 Hyper-V 管理任务和命令的基础组件。

据微软内部人员透露，最终所有的微软图形界面管理工具都会把 PowerShell 作为底层执行命令。例如，您将可以通过底层执行 PowerShell cmdlets 的 Systems Center 系列管理工具，管理 Windows Server role，如 Hyper-V。

在不久的将来，您将完全使用 PowerShell v3 的命令行完成 Hyper-V 的安装和配置工作，不再需要 GUI。

Hyper-V 3.0 的 PowerShell v3

首先从 MSDN 或 TechNet 下载 Windows Server 8 测试版代码。然后，在您选择的 hypervisor 上以虚拟机的方式进行安装。

我的 Hyper-V 3.0 测试环境

我广泛使用 VMware Workstation 和 Fusion，在上面安装测试代码很简单。最新的 VMware Workstation 8 甚至为 Windows Server 8 预装了机器类型，以测试列表的方式列出。我还可以下载到 Windows Server 8 虚拟磁盘——一种定制的.vhd 镜像。然后把它顺利加载到了现有的 Hyper-V 服务器中。

Windows Server 8 安装完成后，通过下面的 PowerShell 命令启动 Hyper-V 角色的安装：

```
Add-WindowsFeature Hyper-V  -Restart
```

（虽然命令中采用了“feature”，但实际上是个 Role）

然后，加载 Hyper-V PowerShell 模块：

```
Import-Module Hyper-V
```

在 Hyper-V 3.0 中运行 PowerShell v3 cmdlets

现在好戏开演。获取所有可以管理 Hyper-V 的 cmdlets 列表：

```
Get-Command  -Module Hyper-V  -Verb
```

向 PowerShell 加载了新模块后，我喜欢使用如下命令列出某个特定对象的所有相关属性：

```
Get-VM | Get-Member  -MemberType Property
```

非常有趣的显示，对吧？现在您可以看到所有的属性，新的和旧的——诸如虚拟磁盘、内存分配、处理器计数等等，现在您可以通过各种 Powershell cmdlets 进行查询。

下面是我找到的一些有趣的新命令，请自行探索更多命令：

```
Enable-VMMigration  #enable migrations
```

该 cmdlets 启用 Hyper-V 主机之间虚拟机的迁移功能

```
Add-VMMigrationNetwork  #add subnets to the VM migration settings
```

通过执行该 cmdlets，可以向 Hyper-V 迁移网络中添加子网。这样可以使支持迁移功能的宿主机设置更为灵活。

```
New-VMReplicationAuthorizationEntry  #add authorization entry for a VM
```

该 cmdlets 创建了新的认证入口，包含新的 Hyper-V 宿主机（或被允许的主服务器），包括在 Hyper-V 集群中对应的副存储。

PowerShell v3 的缺点

当您在 PowerShell 中获得一组新的 Hyper-V cmdlets 同时，也要对其缺点有所了解。新的 Hyper-V 3.0 cmdlets 不能管理老版本的 Hyper-V。这是一个非常严重的问题，意味着管理员根据 Hyper-V 的不同必须采用不同的脚本去管理，直至完成所有宿主机的升级。

在 Hyper-V 3.0 和老版本之间存在不兼容很难理解，也带来不便。但是，大家知道这在软件厂商并不鲜见。对使用老版本 Hyper-V 的管理员，您需要从 CodePlex 下载 PowerShell Library for Hyper-V。

练就火眼金睛 了解优良 PowerShell 函数四大特性

虽然 Windows PowerShell 能以多种方式给管理员带来好处，它的扩展性可能是它最重要的属性。这包括 PowerShell 函数，这些小且简单的任务工具由函数框内的命令组成。运用函数使得调试更简单，也通过让你可以将函数从一个脚本转向另一个来使编写的代码更少。

那么什么时候 PowerShell 函数最有用呢？以下是你需要记住的三个基本指南：

- 如果你一再地重复相同的代码块，例如在电脑上检查多个服务的代码块。在这里，你可以运用函数来执行检查，在所有服务器上运行它，也可以更高效地检测到代码错误。

- 如果你在其它脚本中运用一个专用代码。例如，如果你编写一个递归解析块，你会想要再利用该逻辑。

- 如果该代码在脚本之外有用。这和之前的指南有些许不同，好的例子就是 ping-server 函数。

一般说来，编写代码时考虑再利用常常是个不错的想法，尤其是函数就有再利用的设计时。因此，除违约之外，要考虑函数怎么用、用在哪里能帮助建立它们本该有的参数，这一点很重要。

考虑再利用时，最好是尽最大可能地考虑周全并避免硬编码。此外，编写函数时所有数据都应该通过参数。虽然为参数使用缺省值也说得通，你应该让函数调用者无须修改函数就能指定其它选项。这也是黑箱测试迟早有用的地方，它可以在不同的环境中确定一条函数的可用性。就这一点而言，考虑原始函数的所有变更以及这些变更如何影响脚本整体很重要。举例来说，在 PowerShell V1 中，我常常试着用我自己的转换器执行 verbose 和 whatif 命令。在 V2 中，这个问题已经解决了。

设计函数时你还应该考虑回环和处理逻辑。例如，如果你有处理服务器的逻辑，那么你应该在函数外部维持该逻辑。没有必要为所有函数调用执行它。另一方面，如果你拥有的逻辑明显是函数域，那么你不需要移除它，只要应用调用脚本就行了。

优良 PowerShell 函数的特性是什么？

好的函数都是为特定需求而生然后被弃用，但是所有的好函数都有一些共性。以下是其中的一些特性：

- **优良 PowerShell 函数特性一：定义良好的参数**

一条函数需要非常清楚它预期要返回什么数据。你可以通过应用指定参数来完成该过程。如果你显然有指定值要处理，那么确保它在函数中清晰。完成此过程的一个好方法是通过向要求的$ThisParam 分配参数的缺省值。

- **优良 PowerShell 函数特性二：一致性和预期输出**

你不用猜测函数中会有什么数据，这一点很关键，你宁愿返回的数据是预期的。设计函数所说它返回一种或多种简单数据类型，如线型、日期时间型或布尔运算（Boolean）。但注意，不要用编写输出编写的意外数据或变量中没有抓取的数据污染了这个数据流。

- **优良 PowerShell 函数特性三：独立性**

该函数不应该依赖于来自脚本的任何变量。如果这条函数需要从外部输出，把它变成一个参数。

- **优良 PowerShell 函数特性四：可移植性**

一条函数最简单重要的职能就是可移植性。如果你没有计划要再利用该代码，你也可以编写内联的代码。可移植性的关键因素是确保你的可变名称不会与调用函数冲突。就这一点而言，你可以用$my 或$func 给它们做序。

组策略常用的五大 PowerShell 命令

对于组策略的管理，传统的方式是通过组策略管理控制台（GPMC）来进行。使用图形界面（GUI）让管理员对数千个组策略选项的管理轻松了许多。然而，从 Windows Server 2008 开始，诞生了组策略的 PowerShell 命令，它们可以更加简化组策略的管理过程。

Get-GPO

Get-GPO 命令可以检索到每一个组策略对象（GPO）的所有信息。而且可以根据 GPO 的名称，GPO 的 GUID 来检索组策略信息，也可以使用-all 选项来检索域中的所有组策略。虽然你会觉得通过 GPMC 也能获取这些信息，但是命令的输出还能列出一些通常会错过的信息，如 GPO 的所有者，它的创建时间，最后的修改时间以及启用还是禁用的信息。在网络中对于组策略问题进行排错时，这些信息将至关重要。

Backup/Restore-GPO

虽然可以通过系统状态的备份来对 GPO 进行备份，但是通过一个专门的任务对组策略进行单独备份也是一个不错的主意，毕竟这会让 GPO 的恢复变得更加容易。幸运的是，使用 PowerShell 命令 backup-GPO 就可以做到。与 Get-GPO 协作，可以根据 GPO 的名称，GUID 或者使用-all 选项来指定备份的 GPO。这个命令最有用的部分是可以使用 PowerShell 脚本来定时进行备份：

Backup-Gpo -Name CompanyGPO -Path C:\GPO-Backup -Comment "Monthly Backup"

Restore-GPO 命令可以将 GPO 还原到指定的域。然而，如果你使用 backup-GPO 和 restore-GPO 命令来迁移组策略对象，需要保证 Windows Server2008 操作系统版本的一致性。也就是说，Windows Server 2008 R2 的 GPO 将只能由 Windows Server 2008 R2 进行恢复。

Get-ResultantSetOfPolicy

很久以前，GPMC 就都可以提供组策略的结果报告，这对于组策略的规划和记录来说都是一个非常有用的工具。如果你使用 PowerShell 命令 get-ResultantSetOfPolicy，也可以迅速得到组策略的结果，而且报告可以是 HTML 的

格式。例如，如果你想检查一个特定的用户在特定的计算机上组策略设置的结果，可以运行以下的命令，命令的结果会产生一个所有信息的 HTML 文档：

Get-GPResultantSetofPolicy -user domain\domain.user -reporttype html -path c:\GPO-Reports\UserGPOReport.html

使用所有提到的 cmdlet，PowerShell 还能够提供一种额外的管理能力，那就是将这些命令脚本化，并按照计划执行，这样就可以更有效的监控组策略基础架构的运行状况。

Set/Remove - GPLink

GPLink 的 cmdlet 可以让你创建和删除 GPO 与 OU 之间的关联关系。虽然在 GPMC 中执行这项任务也非常容易，但是 cmdlet 还可以提供另一个方便的管理工具。假如你需要一个 GPO 只是在每个月的某一天运行，其它时间禁止运行。可以在这一天计划运行 GP link 命令将 GPO 和需要的 OU 关联起来，并在这一天结束前将 GPO 的关联删除，整个过程系统自动处理，无需手工运行。你还可以使用其它 GPO 命令与 GPLink 的 cmdlet 进行组合，通过使用管道命令执行 remote-GPLink，以下示例就是如何指定一条组策略或继承组策略，然后删除其链接：

(Get-GPInheritance -Target "ou=CompanyOU,dc=domain,dc=com").GpoLinks | Remove-GPLink

Get-GPPermissions

组策略有时会应用失败的原因之一是因为在 GPO 上不正确的权限设置。Get-GPPermissions cmdlet 会生成详细的报告，报告中会显示 GPO 的访问控制列表（ACL）以及应用的权限。所以，如果你想准确的了解谁对某个 GPO 有权限，可以使用下列命令：

Get-GPPermissions -Name CompanyGPO -TargetName "Company" -TargetType Group

命令会给出以下的输出：

Trustee: Domain Users
TrusteeType: Group
PermissionLevel: GpoRead
Inherited: False

你能看到 ACL 中的所有对象，包括所有的管理组和系统组。在应用 GPO 时，有这样一个简单的输出，就可以非常容易的排除任何基于权限的问题。

Hyper-V 3.0：用 PowerShell 输入输出虚拟机

我最近看到一篇博客提到在 Windows Server 8 中如何通过 PowerShell 启用 Hyper-V，这也让我萌生了一点有关 PowerShell 和 Hyper-V 3 的想法。这篇博客中，我们就一起来看看输入和输出多个虚拟机（Virtual Machine）有多容易。

首先，你可以运行 Get-VM Test*来检查你将输出哪台虚拟机。

```
Get-VM Test*
```

然后，你可以输出该虚拟机：

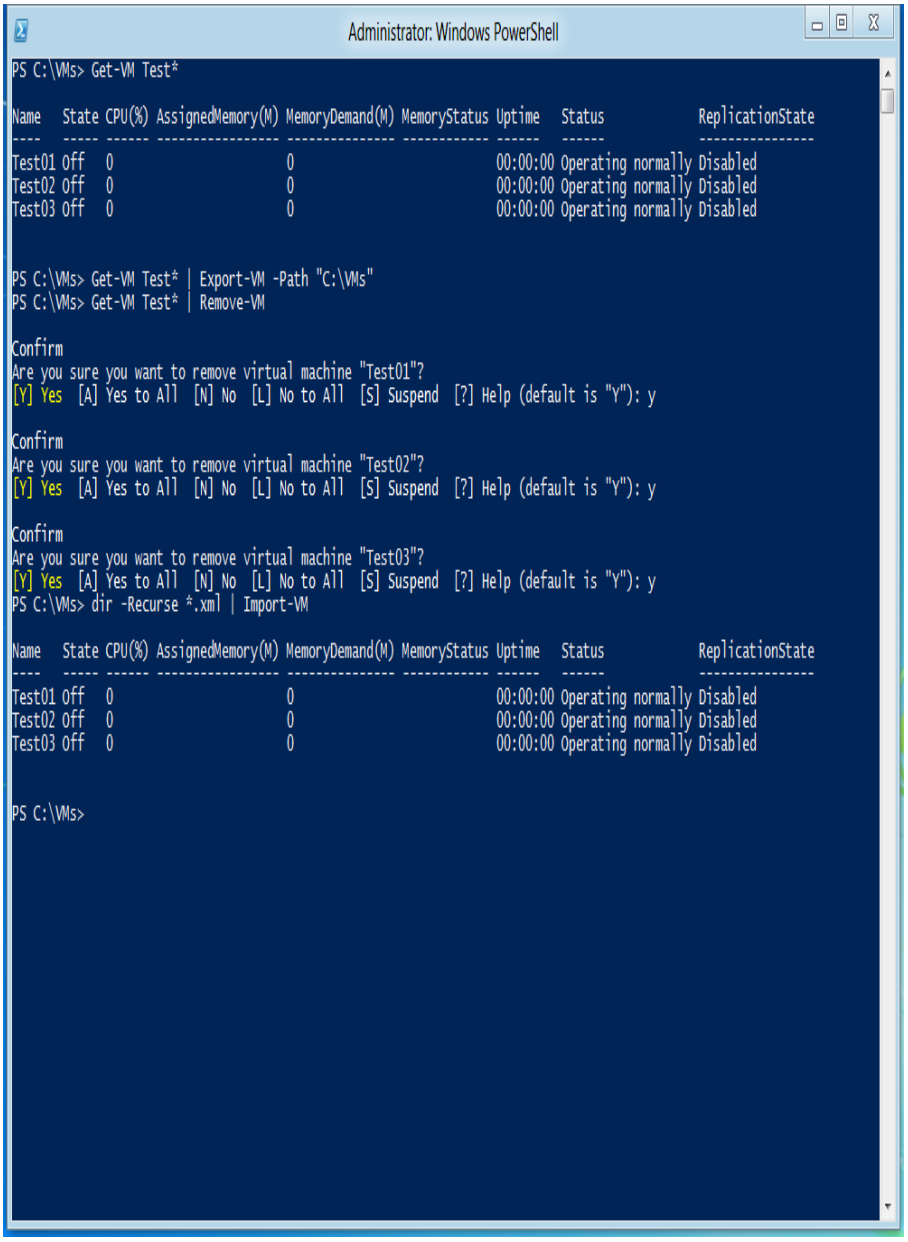
```
Get-VM Test* | Export-VM -Path "C:\VMs"
```

接着，用 PowerShell 在 Hyper-V 中移除该虚拟机（你还可以添加确认参数，这样你就不必为每台 VM 按“y”键）。

```
Get-VM Test* | Remove-VM
```

现在你可以再次输入该虚拟机了：

```
Get-Childitem "C:\VMs" -Recurse *.xml | Import-VM
```


Administrator: Windows PowerShell
PS C:\VMs> Get-VM Test*
Name State CPU(%) AssignedMemory(M) MemoryDemand(M) MemoryStatus Uptime Status ReplicationState
Test01 Off 0 0 00:00:00 Operating normally Disabled
Test02 Off 0 0 00:00:00 Operating normally Disabled
Test03 Off 0 0 00:00:00 Operating normally Disabled
PS C:\VMs> Get-VM Test* | Export-VM -Path "C:\VMs"
PS C:\VMs> Get-VM Test* | Remove-VM
Confirm
Are you sure you want to remove virtual machine "Test01"?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y
Confirm
Are you sure you want to remove virtual machine "Test02"?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y
Confirm
Are you sure you want to remove virtual machine "Test03"?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y
PS C:\VMs> dir -Recurse *.xml | Import-VM
Name State CPU(%) AssignedMemory(M) MemoryDemand(M) MemoryStatus Uptime Status ReplicationState
Test01 Off 0 0 00:00:00 Operating normally Disabled
Test02 Off 0 0 00:00:00 Operating normally Disabled
Test03 Off 0 0 00:00:00 Operating normally Disabled
PS C:\VMs>

用 Hyper-V 3.0 与 PowerShell 管理可扩展交换机

Hyper-V 是微软近年来对虚拟化市场的响应，虽然进展缓慢，但却扎扎实实地提升了微软的 hypervisor 平台。这种进步将在 Windows Server 2012 中持续，下面我们看看跟随微软旗舰服务器操作系统发布的新版 Hyper-V 3.0 有哪些亮点，尤其是如何使用 PowerShell 利用这些功能作完成一些基础任务。

Hyper-V 扩展交换机

Hyper-V 扩展交换机之前未透露，但它对 Windows Server 2012 来说是个神奇的改进。它允许厂商和合作伙伴撰写扩展，并直接插入到 Hyper-V 的网络架构，将虚拟网络的功能从“虚拟端口的面板”扩展为智能的、可管理的、可扩展的虚拟网络硬件。

你能放到交换机中的扩展能完成一系列复杂的或不可能在之前版本的 Hyper-V 和虚拟网络中完成的场景。简言之，微软的合作伙伴已经宣布下个可用扩展将随着 Windows Server 2012 交付给 RTM。

思科发布了为 Hyper-V 打造的 Nexus 1000V，将思科交换机的所有管理和配置功能都进行了打包，并结合了虚拟交换机功能。

5Nine 展示了将服务器用作虚拟防火墙的扩展功能。

Inmon 证明其 sFlow 产品可用于流量捕获与分析。

可扩展交换机有几大优势。其一，获得 Hyper-V 虚拟网络中的新功能，可添加你想要的扩展，还能保持内置虚拟交换机的性能与集成。此外，这些扩展紧密插入，它们继承了 Windows Server 2012 中的所有功能，如热迁移。可直接运行，不需要扩展开发者建立特殊的支持。这些扩展也使用目前的开发架构，并通过验证测试，因此你能基本确定它们运行得很好，不会引发稳定性问题。

PowerShell cmdlets 可用于管理扩展交换机。例如，如果我想要在扩展交换机实例上启用一个具体的扩展，我会使用下面的 Enable-VMSwitchExtension cmdlet 命令。

```
Enable-VMSwitchExtension “name of extension” NameOfSwitchToUse
```

你也可使用 Get-VMSwitchExtension cmdlet 获得可用的扩展交换机目录，如下面的命令：

```
Get-VMSwitchExtension NameofSwitchToUse | fl
```

Hyper-V Replica

Hyper-V Replica 能够让你将一个地点的复虚拟机以及 Hyper-V 和网络连接一同复制到另一个地点。特殊的是，你不需要任何共享存储。在以前，为了在虚拟机之间实现故障转移，你必须设立故障转移群集来访问共享存储如 SAN、NAS 设备或者至少一个 quorum 盘。共享存储不再是必须，本质上意味着虚拟机在主机之间的转移发生在管道中——它们在网络中传达。虚拟机没有中断服务，用户在使用从一台虚拟机复制到另一个主机的工作负载时，将不会看到任何差异。

在以下两种情境中，Hyper-V Replica 功能尤其显著：

- **分散的虚拟机之间的故障转移。**可能订阅服务的顾客故障转移自己的虚拟机到“云”，一个可以在高度互联的数据中心甚至组数据中心支持 Windows Server 2012 的供应者。然后，客户可以直接发送复制信息到云。这使得一个灾难恢复场景中不需要除了 Windows 以外的其他任何软件。

- **Premises-to-premises 支持。**随着需求增长或减弱或操作需求，公司可以在总部和企业园区和较小的分支机构移动主机之间的虚拟机。为了效能靠量、维护窗口或灾难恢复，必要时可以将工作负载从分支机构复制或者复制到分支机构。

你还可以通过 PowerShell 管理 Hyper-V replica。Cmdlets 可以配置复制过程。你得在想要进行复制的主机上开始，并使用 Set-VMReplicationServer cmdlet。

```
Set-VMReplicationServer  -ReplicationEnabled 1  -ComputerName
computertoenable
```

在两台主机上运行这些，然后你可以设置一台虚拟机在其之间进行复制，只要两台服务器是相同的域，只需一个简单的指令 Set-VMReplication：

```
Set-VMReplication  -VMName targetvm  -ReplicaServerName
replicaservertarget  -ReplicaServerPort 80
```

为了加强复制，你可以使用 Start-VMInitialReplication cmdlet 开始复制：

```
Start-VMInitialReplication  -VMName targetvm
```

在 Windows Server 2012 中，Hyper-V 发展成数据云产品。PowerShell 管理此版本中添加的新功能，数据中心管理员会发现日常任务和自动化脚本有了一个全新的水平。

FAQ：使用 Windows PowerShell 命令管理虚拟桌面

对于 IT 专业人士来说，如果他们知道如何使用 PowerShell 的话，那么 PowerShell 就是一款功能非常强大的管理工具。在虚拟桌面环境中，管理员几乎能够像在物理桌面环境下那样使用 Windows PowerShell 命令。

你能够使用 PowerShell 桌面管理 cmdlets 执行基本的任务比如检索虚拟桌面配置数据、构建 PC 池、获取用户信息。对于 VDI 管理员来说，使用命令行监控性能并检查远程活动也是一种很简便的方法。

如果你想知道如何使用 Windows PowerShell 命令、VMware PowerCLI 或 Citrix XenApp cmdlets 管理虚拟和远程桌面的话，那么本文就是个很不错的起点。你还将了解到如何使用脚本和命令管理远程桌面服务（RDS）。

如何使用 PowerShell 管理 Windows 桌面？

Windows 7 和 Windows Server 2008 R2 中内置了 PowerShell，而且 PowerShell 已经成为了微软管理平台的一个基础。如果你知道如何使用 Windows PowerShell 命令管理物理 Windows 桌面的话，那么为什么不使用它来管理虚拟桌面呢？

举例来说，使用一个命令用来获取性能指标，使用另一个命令获取软件清单。对于不熟悉命令行的管理员来说，有些工具能避免很长的命令，甚至还能够帮你输入。

我能够使用哪些命令管理 VMware View？

借助于 VMware 的 PowerCLI 工具，你可以使用 Windows PowerShell 命令管理 vCenter 与 View 之间的连接、创建虚拟桌面池、重建 View 环境甚至还能够执行更多的操作。例如，使用这些 cmdlets 创建手动或者浮动的 View 虚拟桌面池能够帮助你构建协调一致的 View 环境。你可以将 PowerShell 命令行另存为.PS1 文件而且能够很轻松地修改这些文件，但是在输入命令时一定要小心，因为这里面包括了大量的参数。

如何使用 XenApp cmdlets 简化工作任务？

XenApp cmdlets 使你能够更加轻松地管理虚拟化环境、执行日常的管理任务。首先在 Windows 服务器下配置 PowerShell，务必在 XenApp 服务器上安装

cmdlets。然后就可以使用大量的 cmdlets 获取有关 XenApp 虚拟基础设施、应用以及用户的信息。

例如，使用 cmdlet Get-XAFarm cmdlet 获取通用的信息，使用 Get-XAFarmConfiguration cmdlet 获取配置信息。使用 Get-XAServer cmdlets 获取基本的 XenApp 服务器信息，使用 Get-XAServerConfiguration cmdlets 获取详细的服务器信息。

使用 PowerShell 管理远程桌面面临哪些挑战？

在你能够使用 PowerShell 管理远程桌面之前，需要启用 Windows 远程管理服务器并准备好打算管理的远程桌面。为了测试你运行 Windows PowerShell 命令的水平，请输入一个能够返回计算机名的命令。如果能够返回正确的结果，那么就可以在任意一个远程桌面上使用 PowerShell 运行单个或者一连串的命令了。当然你同样可以在内部的虚拟桌面上执行相同的命令。

当心会遇到一些障碍：有时防病毒软件会阻止 PowerShell 的使用，你可能会遇到权限或者防火墙问题。确定你知道远程计算机的 IP 地址，如果遇到身份验证问题，那么你可能必须将远程计算机添加到受信主机列表中。

我能使用 Windows PowerShell 命令管理 XenDesktop 吗？

答案是肯定的，而且你可以使用多达一百多个 PowerShell cmdlets 管理 Citrix XenDesktop。这些命令允许你创建快照，获取管理员账户列表，查看主机、活动目录用户的历史任务，如果你必须断开虚拟桌面会话，那么使用 PowerShell cmdlets 也可以做到。你可以查看 Citrix 的命令参考了解所有的 XenDesktop PowerShell cmdlets。

如何使用 Windows PowerShell 命令管理远程桌面服务？

微软 RDS 内置了 PowerShell 模块，你可以通过开菜单启动 PowerShell RDS 模块，也可以使用 Import-module RemoteDesktopServices-verbose 命令将 PowerShell RDS 模块添加到另一个 PowerShell 模块中。然后就可以使用 Get-command cmdlets 查看可以使用哪些命令管理 RDS 了。

我可以使用哪些有用的命令来管理 RDS？

有一些很棒的命令能够用于管理远程桌面服务。使用 Query Session 命令获取所有 RDS 远程会话的信息，返回结果中包括了活动会话和非活动会话信息。没错你

猜对了，可以使用 Reset Session 命令重置一个会话。Query Uer 命令返回用户信息。另一个重要的命令就是 Change Logon，你可以使用这个命令控制哪些用户能够访问远程桌面服务。

如何使用 Windows PowerShell cmdlets 进行 XenServer 管理

很多管理员发现 Citrix 系统对 Windows PowerShell 支持不好，但 PowerShell 插件可以改善 Citrix XenServer 的管理。

XenServer PowerShell 插件是所有 XenServer 管理员不可缺少的工具，即使您是 XenClient 或 XenServer 命令行的忠实用户。通过向已有 PowerShell 工具箱中添加 PowerShell cmdlets 可以在一个命令窗口下一起管理 Citrix XenDesktop、XenApp、Provisioning Server、Netscaler 和 XenServer。

您可以从 Citrix SDK 网站下载 XenServer PowerShell 插件，我建议同时下载 XenServer 快照插件，如果您的系统借助快照实现备份和容灾的话。

安装 PowerShell 插件

XenServer PowerShell 插件在 32 位系统上的安装非常简单。但如果要在 64 位系统，如 Windows 7 或 Windows Server 2008 R2 上启用，需要运行如下命令完成 DLL 和 64 位.NET Framework 的注册。

C:\windows\microsoft.net\frameowkr64\v2.0.50727\installutil.exe “c:\program files (x86)\citrix\xenserverpssnapin\xenserverpssnapin.dll”

会有信息提示 XenServer PowerShell 注册完成。接下来，把插件添加到 PowerShell 会话或窗口中。如果要检查插件是否已经加载，向 Windows PowerShell 窗口中输入如下命令：

```
if (((Get-PSSnapin -Name "XenServerPSSnapIn" -ErrorAction
SilentlyContinue) -eq $null ) -and ((Get-PSSnapin  -registered -Name
"XenServerPSSnapIn") -ne $null))
{ Add-PSSnapin XenServerPSSnapIn
."C:\Program Files\Citrix\XenServerPSSnapIn\Initialize-Environment.ps1" }
```

现在您已经加载 XenServer PowerShell 插件。运行 Connect-Xenserver 命令以 root 身份登录到宿主机。

PowerShell cmdlets 用于 XenServer 管理

和所有的 Windows PowerShell 模块和插件一样，Get-Command 和 Get-Help cmdlets 是你学习语法和示例的最佳办法。下面是一些我经常使用的 PowerShell cmdlets：

•基于模版创建虚拟机并命名（常用于 POC 测试和实验室环境）

```
oCopy-LocalVM name1 name2 name3 1 3
```

•查看哪些模板可以使用

```
oGet-Template
```

•查看谁是池的所有者：

```
oGet-XenServer:Pool.Master
```

•为虚拟机设置家用服务器

```
oSet-XenServer:VM.Affinity  -VM <name> -Affinity
<xenserver_name>
```

•为宿主机和所有软件授权版本设置授权管理服务器

oSet-XenServer:Host.LicenseServer -server <xenserver> -Host xenserver -LicenseServer <license_host>

oInvole-XenServer:Host.ApplyEdition -server <xenserver_name> -Host xenserver -Edition <enterprise, advanced>

如果要把这些 XenServer PowerShell cmdlets 整合到单一脚本中，您需要创建自己的 XenServer 宿主机和虚拟机自动化环境。

该 PowerShell 脚本是不错的公开脚本示例之一。而 Hypervisor Independent 脚本是可用于 XenServer 和 VMware 的另一个示例。一旦您适应了 XenServer PowerShell cmdlets，通过创建自己的脚本进行 XenServer 宿主机和虚机的创建、修改、汇报和删除工作。

最后还有一点好处就是通过 PowerShell 管理 XenServer，可以收集和报告从 OS 命令行中获得的数据，例如磁盘 I/O 使用情况。幸运的是，XenServer 有集成的 Linux 命令可独立显示宿主机和虚拟机在指定时间段内实际的磁盘 I/O。但该命令对输出数据的格式调整不太好，只简单地输出为文本文件。

幸运的是，Virtualization Jedi 提供了强大的 Windows PowerShell 脚本来阅读这些文本并自动生成 Excel 或 Google Spreadsheets 格式的文件。这样，您就可以收集和分析核心的信息数据。

通过向 XenServer 管理工具箱添加 cmdlets，就可以通过 Windows PowerShell 管理整个系统架构了。

使用 SCVMM PowerShell cmdlets 自定义 Hyper-V 热迁移

对于小的 IT 部门来说，PowerShell cmdlets 强化了 Hyper-V 的热迁移能力。但是，如果能再结合 System Center Virtual Machine Manager 和它的 PowerShell 能力，这些部门就能精确地在不同的节点以及集群共享卷（Cluster Shared Volume)间直接实现热迁移。

Microsoft System Center Virtual Machine Manager (SCVMM) 包括一个强大的图形用户界面（GUI）。管理员们可以利用它来实现 Hyper-V 的热迁移。尽管如此，你可能会觉得 SCVMM 缺少管理一个复杂 Hyper-V 虚拟架构所需的精细控制。

但是，SCVMM PowerShell cmdlets 提供远超过图形界面所提供的更灵活的热迁移管理。通过下列 SCVMM PowerShell cmdlets 和脚本，你可以将整台主机的虚拟机热迁移到某特定的节点，或者基于集群共享卷（Cluster Shared Volumes）的分配来迁移虚拟机。

（备注：你必须在你运行脚本的服务器或工作站上安装 SCVMM 控制台）。

将一个节点上的所有虚拟机迁移到另一个节点

维护模式是一个 SCVMM 图形控制台中的特性，它采用一种智能布局算法来将你选择的节点中的所有虚拟机分布到集群中其它节点。但是如果你希望保持虚拟机的组合，只是想将它们迁移到某一个其它节点上呢？

例如，在我的虚拟架构中，集群中有不同的负载分担应用。将负载均衡的虚拟机工作负载置于一个相同的集群节点上会限制负载均衡的效果，特别是在某台主机故障的情况下。

另外，通过在目标节点保持同样的虚拟机混合，你已经知道对资源的负载会是如何。从我的经验来说，即使是集群中有一个完全空置的节点，SCVMM 的维护模式并不总是将原节点的虚拟机都分布到这个空置的节点上。这是因为 SCVMM 的智能布局算法持续地从各主机收集负载参数来决定最好的虚拟机布局。因此当空置节点接收到虚拟机后，它的布局分数下降，导致 SCVMM 将余下的虚拟机分布到其它的集群节点中。

在图形界面中，唯一能够强制让某个节点的所有虚拟机都热迁移到另一个特定节点的方法是，人工通过每一台虚拟机的迁移向导来实现。但是下面这个脚本能够

覆盖智能布局算法并将虚拟机们同步到一个特定的集群节点上。你只需要很简单的回答几个提示。

```
# ----------------------------------------------------------------------
--------
# Migrate All VMs to Alternate Node in the Same Cluster
# ----------------------------------------------------------------------
--------
Add-PSSnapin Microsoft.SystemCenter.VirtualMachineManager
$VMM = Read-Host "Enter Name of VMM Server"
$SH = Read-Host "Enter Source Host Name"
$DH = Read-Host "Enter Destination Host Name"
Get-VMMServer -computername $VMM
Get-VM | ? {$_.hostname -like "$SH*"} | Move-VM -VMHost "$DH*"
```

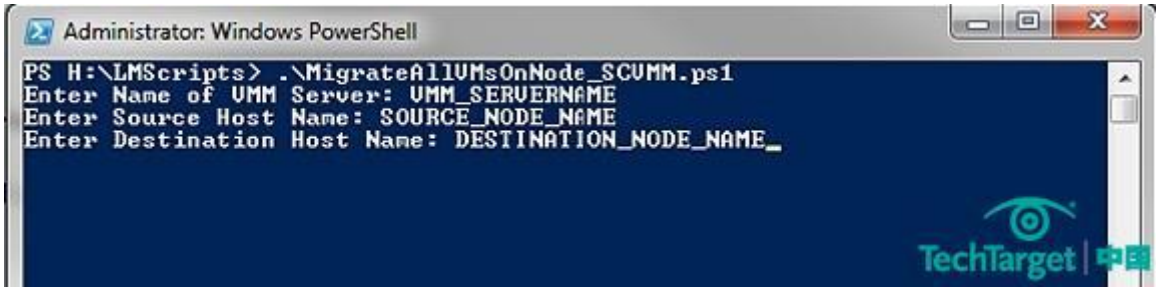
通过以下步骤来运行这个脚本：

1.将上面的 SCVMM PowerShell 脚本保存下来（例如保存 MigrateAllVMsOnNode_SCVMM.ps1）；

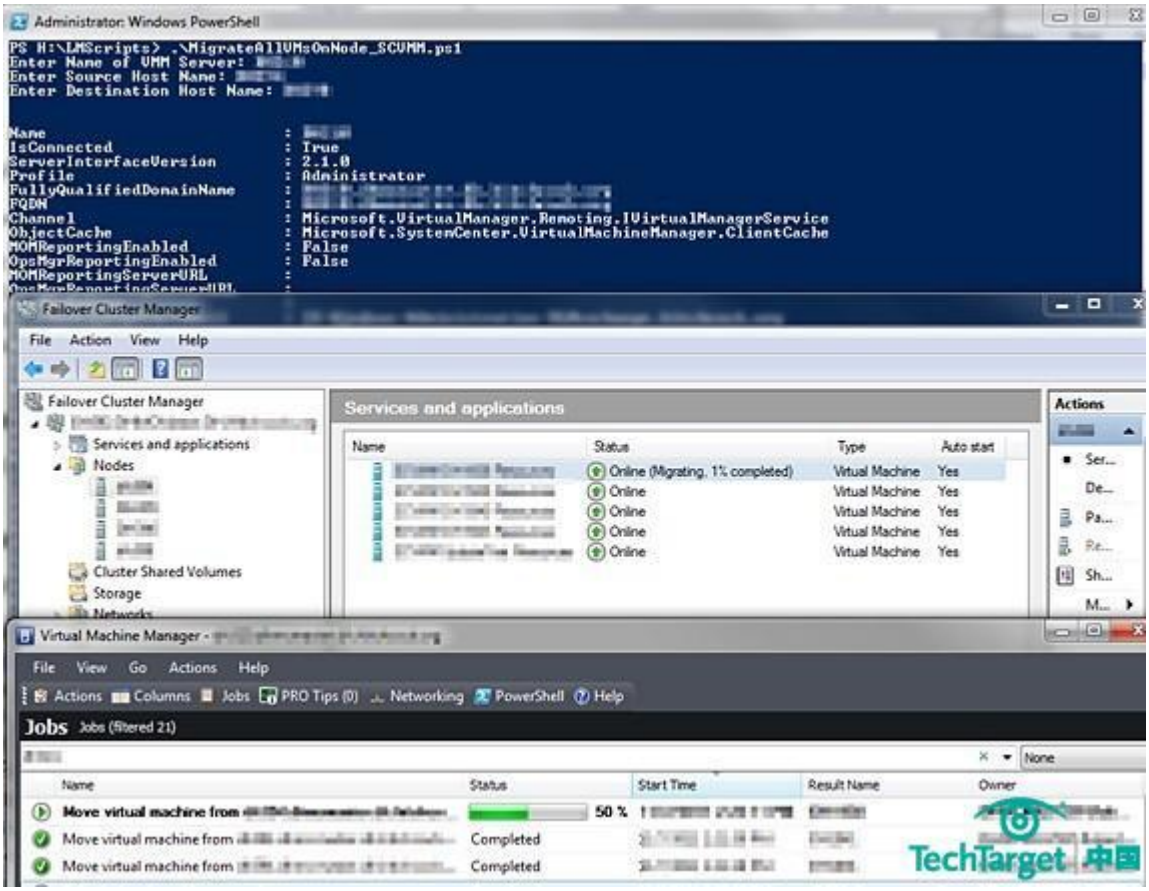
2.打开 Windows PowerShell；

3.运行脚本；

4.回答对于 SCVMM 服务器、源节点名以及同一个集群中的目标节点名的提示。

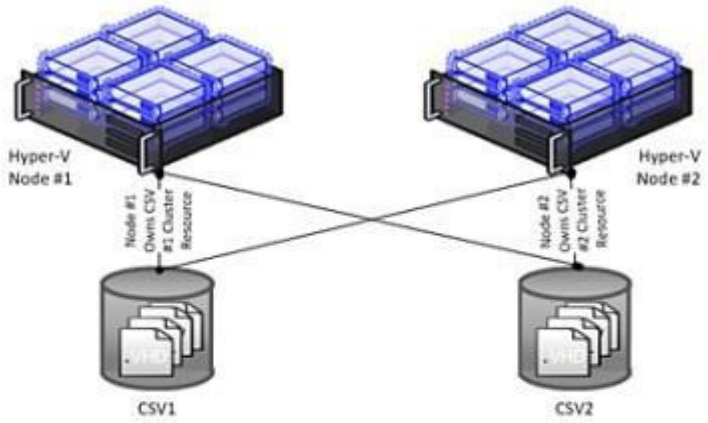


5.通过观察命令状态，Failover Cluster Manager 或者 SCVMM 的任务页来监视迁移进度。



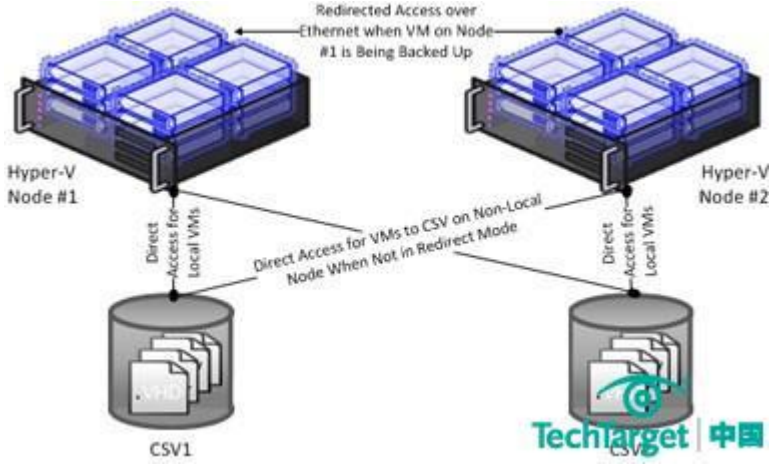
在节点间对于一个集群共享卷（Cluster Shared Volume，CSV）上的虚拟机进行迁移

这个脚本识别一个特定集群共享卷上的虚拟机，并将他们热迁移到某特定的目标节点。使用这个脚本来将一个集群共享卷上的虚拟机在热迁移后保持在同一台主机上。



Architecture Recommendation: To avoid redirected access performance degredation during backups using the Hyper-V software VSS Writer, Node #1 owns CSV #1 cluster resource and VMs on Node #1 have files on CSV #1. Node #2 owns CSV #2 cluster resource and VMs on Node #2 have files on CSV #2.

你为什么会需要这样做？在通常的集群操作中，一个虚拟机可以直接访问一个被很多节点共享的卷，并且那个集群中的其它节点也能同时利用这个节点的资源。但是当你使用 Microsoft Data Protection Manager 或者其它基于 Hyper-V 卷影副本(VSS)的备份工具时，问题就出来了。在备份的过程中，只有拥有这台虚拟机的主机能够直接访问磁盘。处于其它节点中，并共享该相同的集群共享卷的虚拟机，则需要通过网络来访问磁盘，这扩大了磁盘的延时以及降低了性能。



为了避免这个情况，指定下列的布局架构来对所有共享 CSV 的虚拟机都维持全性能的磁盘访问。在使用维护模式后，通过使用这个脚本来轻松地将某个 CSV 上的虚拟机都热迁移到所需的节点上

```
# ------------------------------------------------------------------------
--------
# Live Migrate Virtual Machines On a Particular Volume to a New Host in
Same Cluster
# ------------------------------------------------------------------------
--------
Add-PSSnapin Microsoft.SystemCenter.VirtualMachineManager
$VMM = Read-Host "Enter Name of VMM Server"
$SH = Read-Host "Enter Source Host Name"
$DH = Read-Host "Enter Destination Host Name"
$Vol = Read-Host "Enter Volume/CSV Volume to Move VMs to Destination
Host"
Get-VMMServer -computername $VMM
Get-VM | ? {$_.hostname -like "$SH*"} | ? {$_.Location -like "*$Vol*"} |
Move-VM -VMHost "$DH*"
```

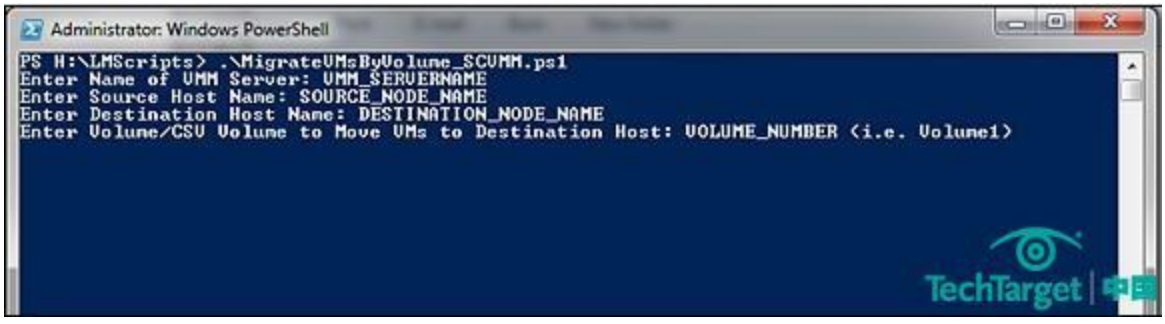
通过下列步骤来运行这个脚本：

1.将上面的脚本保存（例如 MigrateAllVMsByVolume_SCVMM.ps1）；

2.打开 Windows PowerShell；

3.运行你保存的上述脚本；

4.回答对于 SCVMM 服务器、源节点名、目标节点名以及你想指向的集群共享卷（CSV）。



5.通过观察命令状态，Failover Cluster Manager 或者 SCVMM 的任务页来监视迁移进度。

对于 SCVMM PowerShell cmdlets 进一步的实验

上面的脚本仅仅是两个你能使用 SCVMM PowerShell cmdlets 的例子。由于 SCVMM 深度采集每台虚拟机的信息，有更多的方法能够将特定虚拟机包括或者排除在热迁移中：

- **名字**：你可以对于某个特定的名字属性来热迁移虚拟机（例如使用-like 命令选项）；
- **内存**：你可以指向处于某内存临界之上或之下的虚拟机；
- **操作系统**：你可以通过操作系统来选择虚拟机，例如 Windows、Linux（仅快速迁移）等。

通过 SCVMM PowerShell cmdlets，你能够较采用图形控制台更深度地定制你的热迁移方案。System Center Virtual Machine Manager 2012 会在升级的 cmdlet 中添加更多的选项。但是图形界面仍将缺少精确的管理功能。因此，学会使用 SCVMM PowerShell cmdlets 来简化管理任务，会对你受益匪浅

使用 XenServer PowerShell cmdlets 开发高级脚本

没有 XenServer Powershell cmdlets 的 Critix 管理工具箱是不完整的。XenServer Powershell cmdlets 允许管理员使用单个命令行接口管理 Citrix XenDesktop、XenApp、部署服务器、Netscaler 以及 XenServer。

在虚拟环境中，基本的 XenServer PowerShell cmdlets 提供了一些关键的特性，允许管理员拷贝虚拟机，判定主资源池，配置许可、主机版本并为虚拟机分配服务器。但是虚拟环境只是个基础；管理员通常需要使用高级脚本以及程序进一步将任务序列化并自动化。

开始使用 XenServer PowerShell cmdlets

为了在 Window 7 x64 服务器上安装 XenServer PowerShell cmdlets，管理员需要下载 XenServer PowerShell 管理单元并提取 cmdlets。为了在 x64 系统中配置管理单元，需要在命令提示符下运行如下命令，使用 64 位的.NET 框架注册动态链接库。

```
C:\windows\microsoft.net\framework64\v2.0.50727\installutil.exe
"c:\program files (x86)\citrix\xenserverpssnapin\xenserverpssnapin.dll"
```

运行完上述命令后，应该会收到一条消息，显示命令提交成功。

如果你在使用快照，那么你还需要使用 XenServer 快照管理单元。确保将快照管理单元增加到 PowerShell 配置文件中并且不要忘了执行 Connect-Xenserver 命令启动会话。否则在 PowerShell 下将会提示错误信息。

使用 MAC 地址查找 XenServer 客户端

为了通过 MAC 地址识别虚拟机，你必须使用如下命令获取虚拟接口（VIF）的对象属性：

```
$vif = get-xenserver:VIF | ? { $_.MAC -match
"ce:e2:b7:56:85:7f"};(Get-XenServer:VIF.VM -VIF $vif.uuid).name_label
```

在 Get-Xenserver cmdlet 中应用 VIF 属性直接查询虚拟接口的 MAC 地址。以下命令说明了如何检索特定 UUID（通用唯一标识符）的虚拟机信息。使用该信息

替代 VIF 属性。只需要改变不同的属性值，就能够基于这些属性值检索、管理并关联不同的结果。

```
Get-XenServer:VM -properties @{ uuid='$uuid' }
```

令人称道的是 Get-XenServer cmdlets 提供了一系列可用的变量，包括角色，资源池，秘密，会话以及契约。运行 get-help Get-XenServer 命令可以查看所有可用的变量值并快速创建你自己的查询。

除 Get-XenServer 之外的其他 cmdlet

除了经常用到的 Get-XenServer cmdlet 之外，还有一些其他的 cmdlets。你可以使用 XenServer 提供的功能，比如创建、添加、删除、回退、设置以及移除。

为了查看可用的 XenServer cmdlets 及其属性，可以在 PowerShell 命令提示符下运行 Get-Command *xen* 命令。

我发现了一些能够展示 XenServer cmdlets 内部工作机制的高级脚本。这些高级脚本来源于 Maikel Gadder 的博客，它提供了一些非常好的如何使用变量并存储属性的例子。

```
# Xenserver-UpdateScript

# V 0.9.2

# 20.07.2012

# written by Maikel Gaedker (http://www.ingmarverheij.com/ #
Version : 1.0, 13 february 2012 # # Requires : plink (a command-line
interface to the puTTY back ends) #
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

# function StartProcess([String]$FileName,
[String]$Arguments){ $process = New-Object "System.Diagnostics.Process"
$startinfo = New-Object "System.Diagnostics.ProcessStartInfo"
$startinfo.FileName = $FileName $startinfo.Arguments = $arguments
$startinfo.UseShellExecute = $false

$startinfo.RedirectStandardInput = $true
```

```
$startinfo.RedirectStandardOutput = $true $startinfo.WindowStyle =
[System.Diagnostics.ProcessWindowStyle]::Hidden

$process.StartInfo = $startinfo

$temp = $process.start() return $process } #Region PrerequisiteCheck
#Check number of arguments If ($args.count -lt 5) { Write-Host "Usage"
Write-Host "powershell.exe .\RDSConnect.ps1 (XenServerPoolMaster)
(XenServerUsername) (XenServerPassword) (VMName) (Network ID)
[CustomFieldName] [CustomFieldValue]" Write-Host ""

Write-Host "Example"

Write-Host "powershell.exe .\RDSConnect.ps1 172.16.1.1 root PasswOrd
WS01 0 STUDENT 1"

Write-Host "" Write-Host "Press any key to continue ..."

$x = $host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")

break
}

    #EndRegion

    #Region Define variables and read

#Executables

$strExecutableMSTSC=((Get-Item
"Env:WinDir").Value)+'\system32\mstsc.exe' $strExecutablePLink=(Split-
Path -parent $MyInvocation.MyCommand.Definition) + '\plink.exe' #File
paths $strPathTemp=$Env:TEMP

$strFileQueryNetworks ='QueryNetworks'

#Script variables $XenServerHost=$args[0] $XenServerUsername=$args[1]
$XenServerPassword=$args[2] $VirtualMachineName=$args[3]
```

```
   $VirtualMachineNetworkID=$args[4].toString()

    If ($args.count -ge 7) {

$CustomFieldName=$args[5] $CustomFieldValue=$args[6] } else
{ $CustomFieldName=""

$CustomFieldValue="" } $strNICInterface=($VirtualMachineNetworkID)+'/ip:
' #Filter variables

    $strFilterVM='name-label="' + $VirtualMachineName+'"'

    IF ($CustomFieldName) {$strFilterVM+=' other-
config:XenCenter.CustomFields.' + $CustomFieldName + '=' +
$CustomFieldValue}

    #EndRegion #Prevent rsa2 key fingerprint message
#================================== #The server's host key is not
cached in the registry. You have no guarantee that the server is the
computer you #think it is. #The server's rsa2 key fingerprint is: ssh-
rsa 2048 7c:99:f3:31:38:ca:b7:b6:3b:21:53:55:ff:f3:76:1e #If you trust
this host, enter "y" to add the key to PuTTY's cache and carry on
connecting.

    #If you want to carry on connecting just once, without adding the
key to the cache, enter "n".

    #If you do not trust this host, press Return to abandon the
connection. # #Run plink and confirm rsa2 key fingerprint with yes

    #------------------------------------------------ $process =
StartProcess $strExecutablePLink (' -l '+$XenServerUsername+' -pw
'+$XenServerPassword+' '+$XenServerHost+' exit')

    $process.StandardInput.WriteLine('y') #Retrieve IP of VM
#================ # #Create a script to query a XenServer and ask the
networks of the VM #-----------------------------------------------------
--------------- New-Item $strPathTemp -Name $strFileQueryNetworks -type
file -Force | Out-Null Add-Content ($strPathTemp + '\' +
```

```
$strFileQueryNetworks) -Value ('xe vm-list '+$strFilterVM+' os-
version:distro="windows" params=networks --minimal')

    #Run the script on the specified XenServer

    #---------------------------------------- $process = StartProcess
$strExecutablePLink (' -l '+$XenServerUsername+' -pw
'+$XenServerPassword+' '+$XenServerHost+' -m '+($strPathTemp + '\' +
$strFileQueryNetworks))

    $VMNetworks = $process.StandardOutput.ReadLine()

    Remove-Item ($strPathTemp+'\'+$strFileQueryNetworks)

    #Determine if the networks of the virtual machine can be found

    #-------------------------------------------------------------
if(!$VMNetworks) { Write-Host "The virtual machine
'"$VirtualMachineName"' could not be found." Write-Host "" Write-Host
"Press any key to continue ..." $x =
$host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")

      break } else {

      #Determine the IP address of the NIC can be found foreach
($strVMNetwork in $VMNetworks.Split(";")) { if
($strVMNetwork.Contains($strNICInterface))
{ $strVMIPaddress=$strVMNetwork.Substring($strVMNetwork.IndexOf($strNICI
nterface) + $strNICInterface.Length) } }

      #Determine if the IP address of the network ID can be found #-----
-------------------------------------------------------
if(!$strVMIPaddress) { Write-Host "The IP address of network
'"$VirtualMachineNetworkID"' could not be found." Write-Host "" Write-
Host "Press any key to continue ..." $x =
$host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown") break

    } else { Write-Host "The virtual machine '"$VirtualMachineName"' is
connected via IP "$strVMIPaddress }
```

```
}

#Start MSTSC to the VM

#======================

$processMSTSC=StartProcess $strExecutableMSTSC
('/v:'+$strVMIPaddress+' /f')
```

最后，在 Citrix 博客中，Ajene' Hall Barrett 贴出了一款功能非常强大的脚本，实现了虚拟桌面的自动化部署。这个脚本包括了变量引用，活动目录 cmdlets，以及 XenServer 和 PVS（部署服务器）cmdlets。所有的脚本编写良好，而且非常有参考价值。

高级脚本进一步证明了你可以使用 XenServer PowerShell cmdlets 实现自动化管理。随着 Citrix 工具集的不断发展，所有的这些功能应该会进一步扩展并增强。